ALLQBF Solving by Computational Learning[⋆]

Bernd Becker[1], Rüdiger Ehlers[2], Matthew Lewis[1], and Paolo Marin[1]

[1] Albert-Ludwigs-Universität Freiburg
[2] Universität des Saarlandes

Abstract. In the last years, search-based QBF solvers have become essential for many applications in the formal methods domain. The exploitation of their reasoning efficiency has however been restricted to applications in which a "satisfiable/unsatisfiable" answer or **one** model of an open quantified Boolean formula suffices as an outcome, whereas applications in which a compact representation of **all** models is required could not be tackled with QBF solvers so far.
In this paper, we describe how computational learning provides a remedy to this problem. Our algorithms employ a search-based QBF solver and learn the set of all models of a given open QBF problem in a CNF (conjunctive normal form), DNF (disjunctive normal form), or CDNF (conjunction of DNFs) representation. We evaluate our approach experimentally using benchmarks from synthesis of finite-state systems from temporal logic and monitor computation.

Keywords: QBF, Computational learning, QBF model enumeration

1 Introduction

Recent progress in quantified Boolean formula (QBF) and satisfiability (SAT) solving has strengthened the applicability of such solvers in many areas of formal methods. For example, in *bounded model checking* [5], the question whether some property holds along a run of a given system with some bounded length is encoded into a SAT formula, and then subsequently solved. In case the formula is found to be satisfiable, from a corresponding assignment to the variables, we can reconstruct a run that violates the specification. When generalizing from SAT to QBF solving, we can use the universal quantifiers to apply a more concise problem encoding or ask more complex questions such as: "do there exist values for some parameters in a system such that for every input of some fixed length to the system, we do not reach some error state?". In case of a positive answer, it is desirable to obtain values for the parameters. This is called *open QBF* solving, as here, we leave some variables in the QBF instance unquantified and ask for an assignment to these variables that witness the satisfiability of the QBF formula. Such an assignment is also called a *model* of the formula.

For other applications, however, obtaining one model is not enough, but we rather need *all models* of an open QBF formula. Representatives of this class are the synthesis of finite-state systems from temporal logic specifications, which is frequently reduced to game solving, and building a system monitor for identifying that a system

⋆ This work was partly supported by the DFG as part of the AVACS Transregional Collaborative Research Center (SFB/TR 14).

reached a potentially bad state, i.e., a state from which some error state can be reached within a short amount of time. As in these applications, there can easily be millions or even billions of models for an open QBF formula, it is furthermore not sufficient to just enumerate the models, but we rather need a *compact representation* of them. Resolution-based *variable elimination* techniques are known not to scale well for many variables to be eliminated, so we need some alternative approach for obtaining such a compact representation of all models of a given SAT or QBF instance, which we call the *ALLSAT* and *ALLQBF* problems for the scope of this paper.

For the ALLSAT problem, some solutions that go beyond simple model enumeration and successive variable elimination are known. A typical ingredient of such an approach is the enumeration of solution cubes [7,21], which leads to a DNF representation of the set of models, possibly combined with some on-the-fly or a-posteriori post-processing to obtain a CNF representation [7,8] of the model set. For ALLQBF, search-based solving approaches that go beyond simple model enumeration [3] and variable elimination by resolution are unknown so far. Thus, for applications in which the ALLQBF problem has to be solved for instances that have many models, but for which there are also many quantified variables, no feasible solution exists yet.

In this paper, we present an approach to extend a state-of-the-art search-based QBF solver to an ALLQBF solver by employing *(active) computational learning* [24], which is the process of deriving a model of some data by asking questions to some *teacher oracle*. Computational learning should not be confused with *clause learning*, a technique to increase the performance of SAT and QBF solvers.

Our approach learns a CNF (conjunctive normal form), DNF (disjunctive normal form) or CDNF (conjunction of DNFs) representation of the set of all models of an open quantified Boolean formula, i.e., a QBF instance in which some variables are left free. The algorithms for all of these result types can equally be applied for ALLSAT solving, but the main target of our approach is ALLQBF solving. Benchmarks from synthesis of finite-state systems and monitor generation show the effectiveness of the new approach.

We start by stating the required preliminaries in Section 2 and describe our approach to learn DNF, CNF or CDNF representations of the set of models of an open QBF problem in Section 3. Section 4 discusses how a modern QBF solver can be adapted to its use as oracle in a learning process. In Section 5, we discuss *synthesis* and *monitor generation* as two of the application of ALLQBF solving, from which we take the Benchmarks for our experimental evaluation of a prototype implementation of our learning approach in Section 6. We conclude with a summary.

2 Preliminaries

In this paper, we consider open quantified Boolean formulas (QBF) in *prenex-cnf-form*, i.e., for a finite set of *free variables* V, we define the set of QBF instances $\mathcal{Q}(V)$ over V as all formulas of the type:

$$\psi = Q_1 x_1 . Q_2 x_2 . \ldots Q_n x_n . \phi \tag{1}$$

where $n \in \mathbb{N}$, $Q_i \in \{\forall, \exists\}$ for all $1 \leq i \leq n$, and ϕ is a Boolean formula in conjunctive normal form over the set of variables $\{x_1, \ldots, x_n\} \cup V$. A Boolean formula is said to be in *conjunctive normal form* if it is a conjunction of a set of *clauses* $\phi = \bigwedge_v C_v$. A clause C_v is in turn a set of literals that is treated disjunctively, i.e., $C_v = \bigvee_w l_w$. A *literal* is a variable or its negation. In (1), $Q_1 x_1.Q_2 x_2. \dots Q_n x_n$ is the *prefix* and ϕ is the *matrix*. The *level of a variable* x_i is defined to be one plus the number of expressions $Q_j x_j.Q_{j+1} x_{j+1}$ in the prefix with $j \geq i$ and $Q_j \neq Q_{j+1}$ (plus 1 if $Q_1 = \forall$). For the sake of simplicity, we will use the term *outermost* (respectively *innermost*) quantifier level to indicate variables having the highest (respectively lowest) level. The level of a literal is the level of its variable. A literal l is *universal* if $l = v_i$ or $l = \neg v_i$ for some $1 \leq i \leq n$ with $Q_i = \forall$. All other literals are *existential*. In (1), a literal l is

- *unit* if l is existential, and, for some $m \geq 0$,
  - a clause $(l \vee l_1 \vee \ldots \vee l_m)$ is a clause in ϕ, and
  - each literal l_i $(1 \leq i \leq m)$ is universal and has a level lower than l.
- *monotone* or *pure* if
  - either l is existential, $\neg l$ does not occur in any clause in ϕ, and l occurs in some clauses in ϕ;
  - or l is universal, l does not occur in any clause in ϕ, and $\neg l$ occurs in some clauses in ϕ.
- *don't care* if l is existential, and neither l nor $\neg l$ occur in any clause in ϕ.

Any element of $\mathcal{Q}(V)$ can be seen as a function that maps some variable valuation $f \in (V \rightarrow \mathbb{B})$ to either $\mathbf{true}$ or $\mathbf{false}$. If $Q_1 = \forall$, we say that ψ is outermost universally quantified, and if $Q_1 = \exists$, we say that ψ is outermost existentially quantified. We call $\mathcal{Q}(\emptyset)$ the set of *closed QBF instances*. For any set V, $\mathcal{Q}(V)$ is the set of *open QBF instances* over V.

Given some set of variables V and some open QBF formula $\psi \in \mathcal{Q}(V)$, we say that some variable valuation $f \in (V \rightarrow \mathbb{B})$ is a model of ψ if $\psi(f) = \mathbf{true}$. Likewise, f is a *co-model* of ψ if $\psi(f) = \mathbf{false}$. A *partial variable valuation* is a function $f' : V \rightarrow \{\mathbf{false}, \mathbf{true}, \perp\}$, and a variable valuation f is a *completion* of f' if for every $v \in V$, $f'(v) = f(v)$ if $f'(v) \neq \perp$. We say that a partial variable valuation is a *partial model* (*partial co-model*) of some open QBF formula ψ if every completion of the partial valuation is a model (co-model) of ψ.

We call a conjunction of literals a *term* and a disjunction of terms a Boolean formula in *disjunctive normal form*. For a partial (or complete) variable valuation $f' \in (V \rightarrow \{\mathbf{false}, \mathbf{true}, \perp\})$, we define the term induced by f' as follows:

$$\mathrm{term}(f') = \Big(\bigwedge_{v \in V, f'(v) = \mathbf{true}} v \Big) \wedge \Big(\bigwedge_{v \in V, f'(v) = \mathbf{false}} \neg v \Big)$$

For some Boolean formula in disjunctive normal form $t_1 \vee t_2 \vee \ldots t_m$, we define $\mathrm{terms}(t_1 \vee t_2 \vee \ldots \vee t_m) = \{t_1, t_2, \ldots, t_m\}$. Likewise, for some term $t = l_1 \wedge l_2 \wedge \ldots \wedge l_m$, we define $\mathrm{lits}(t) = \{l_1, l_2, \ldots, l_m\}$. For a Boolean formula ψ over some set of variables V, some term $t = l_1 \wedge \ldots \wedge l_m$ is an implicant of ψ if $\neg t \vee \psi \equiv \mathbf{true}$. A term is called a *prime implicant* if we cannot remove any of its literals without losing the property that it is an implicant. A Boolean function $F : (V \rightarrow \mathbb{B}) \rightarrow \mathbb{B}$ is called monotone if for every $f, f' : V \rightarrow \mathbb{B}$ with $F(f) = \mathbf{true}$ and for all $v \in V$, $f(v) = \mathbf{true}$ implies $f'(v) = \mathbf{true}$, we have $F(f') = \mathbf{true}$. We denote the *exclusive or* Boolean operator by $\oplus$ and the function composition operator by $\circ$.

3 ALLQBF Solving by Computational Learning

In this section, we show how to use computational learning techniques to obtain compact representations of the sets of models of open QBF formulas that are specified in the commonly used prenex-cnf-form.

Given an open QBF formula ψ over the set of variables V, we consider three representations for a set of models of ψ here: a disjunctive normal form (DNF) Boolean formula, a conjunctive normal form (CNF) Boolean formula, and a conjunction of DNFs (called CDNF), all over V. We start this section by first declaring the requirements to the QBF solver that we use as an oracle in the following learning algorithms, and then explain how to compute a DNF, CNF or CDNF representation of the set of models using the QBF solver as oracle in Sections 3.2, 3.3, and 3.4, respectively.

3.1 Requirements to the QBF Solver Used

For the henceforth algorithms, we use a QBF solver for open QBF formulas in prenex-cnf-form as an oracle in the learning process, and apply it for checking the satisfiability and non-universality of open QBF formulas. In case of a positive answer to one of these checks, the solver must be able to return a (partial) model/co-model of the open QBF formula, respectively, i.e., a valuation for the unquantified variables.

In the experimental evaluation of the following learning schemes, we used an open-QBF version of QuBE 7.2 [15]. The modifications that were required are more complex than one might think, as QuBE 7.2 uses a very advanced preprocessor that can remove and rename variables. This preprocessor allows it to achieve high levels of performance, but it can make the production of models and co-models nontrivial. So, the modifications for satisfying the requirements above, while still using the preprocessor, are described in Section 4.

3.2 Learning DNFs

Let $\psi = Q_1 x_1 . Q_2 x_2 . \ldots Q_n x_n . \phi$ be an open QBF formula over the free variables V such that ϕ is in conjunctive normal form. For learning a DNF representation of the set of models of ψ, we apply a variant of the classical algorithms from learning theory for obtaining monotone DNFs [2] or k-term DNFs [17] from a function to learn. Our variant, depicted in Algorithm 1, makes use of the fact that when learning from open QBF formulas, we can take advantage of the possibility to check if a term is an implicant of ψ.

In the algorithm, terms are repeatedly added to the *candidate* DNF representation ψ' until ψ' represents precisely the set of models of ψ. In line 2 of the algorithm, it is checked using a QBF solver as an oracle whether there exists some variable valuation to the variables in V that is a model of ψ, but not yet of ψ'. Whenever this is the case, we know that ψ' is not yet complete and search for a prime implicant of ψ' that also implies the newly found variable valuation.

Note that since the negation of a DNF formula is a CNF formula, $(\phi \wedge \neg\psi')$ is in CNF, and thus the QBF instance ρ computed in line 2 is in prenex-cnf-form. The algorithm uses the partial model of ρ as a starting point for finding the next prime implicant.

Of course, the model can also be complete, but does not need to be. In particular, if the QBF solver finds a satisfying assignment to V while still preprocessing, the model will typically be incomplete.

In the remaining lines of the algorithm, the implicant f' (represented in form of a partial variable valuation) obtained in line 3 is reduced as much as possible in order to obtain a prime implicant. In line 7, we take profit from the fact that we use a QBF solver as oracle: by universally quantifying over the variables that are not set in f' and one additional variable v, we can check whether $f'(v)$ can be set to $\perp$ without changing the fact that f' is an implicant. The algorithm guarantees that at the end of

Algorithm 1: DNF learning using a QBF solver as oracle

```
   Data: An open QBF instance $\psi = Q_1 x_1 . Q_2 x_2 . \dots Q_n x_n . \phi$ over the set of variables $V$
   Result: The set of its models represented as DNF $\psi'$
   begin
1      $\psi'$ := false
2      while $\rho := Q_1 x_1 . Q_2 x_2 . \dots Q_n x_n . (\phi \wedge \neg\psi')$ is satisfiable do
3          $f'$ := partial model of $\rho$
4          for $v \in V$ do
5              if $f'(v) \neq \perp$ then
6                  $f'' := f' \setminus (v \mapsto f'(v)) \cup (v \mapsto \perp)$
7                  if $\forall \{v' \in V \mid f''(v') = \perp\} . Q_1 x_1 . Q_2 x_2 . \dots Q_n x_n . (\phi \wedge \mathrm{term}(f'')) \not\equiv$ false
                   then
8                      $f' := f''$
9          $L := \{\neg v \mid v \in V, f'(v) = \mathbf{false}\} \cup \{v \mid v \in V, f'(v) = \mathbf{true}\}$
10         $\psi' := \psi' \vee \bigwedge_{l \in L} l$
   end
```

the computation, the DNF representation of the set of models of ψ is *irreducible*, i.e., there are no superfluous literals or terms in the obtained DNF ψ'.

3.3 Learning CNFs

Learning CNFs instead of DNFs as described above can be seen as the dual case. Since for search-based QBF solving, the matrix of a Boolean formula has to be in CNF, we would however have to re-encode this matrix to complement the original formula. Thus, we apply a slightly different method, aiming to skip the re-encoding step into prenex-normal-form of the formula.

Algorithm 2 describes the modified procedure. In this algorithm, the function Enc^- is used to map a Boolean formula in CNF form into a CNF formula that encodes its negation (using a Tseitin encoding [23]), and EncV^- describes the necessary variables. Let $\psi = \bigwedge_{1 \leq i \leq m} \bigvee_{1 \leq j \leq k_i} l_{ij}$ be a CNF formula with $m \in \mathbb{N}$, $k_1, \dots, k_m \in \mathbb{N}$, and

Algorithm 2: CNF learning using a QBF solver as oracle

```
Data: An open QBF instance $\psi = Q_1 x_1.Q_2 x_2.\dots Q_n x_n.\phi$ over the set of variables $V$
Result: The set of its models represented as CNF $\psi'$
begin
1   $\psi' := \mathbf{true}$
2   while $\rho := \exists s, \mathsf{EncV}^-(\psi').Q_1 x_1.Q_2 x_2.\dots Q_n x_n.((\phi \vee s) \wedge (\mathsf{Enc}^-(\psi') \vee \neg s))$ is
    non-universal do
3       $f' :=$ partial co-model of $\rho$
4       for $v \in V$ do
5           if $f'(v) \neq \bot$ then
6               $f'' := f' \setminus (v \mapsto f'(v)) \cup (v \mapsto \bot)$
7               if $Q_1 x_1.Q_2 x_2.\dots Q_n x_n.(\phi \wedge \mathrm{term}(f'')) \equiv \mathbf{false}$ then
8                   $f' := f''$
9       $\psi' := \psi' \wedge \left(\bigvee_{v \in V, f'(v)=\mathbf{false}} v \vee \bigvee_{v \in V, f'(v)=\mathbf{true}} \neg v\right)$
end
```

$l_{ij} \in V \cup \{\neg v \mid v \in V\}$ for $1 \leq i \leq m$ and $1 \leq j \leq k_i$. We define:

$$\mathsf{EncV}^-(\psi) = \{v_i \mid 1 \leq i \leq m\}$$
$$\mathsf{Enc}^-(\psi) = \Big(\bigvee_{1 \leq i \leq m} v_i\Big) \wedge \bigwedge_{1 \leq i \leq m} \bigwedge_{1 \leq j \leq k_i} (\neg v_i \vee \neg l_{ij})$$

The algorithm is based on the idea to iterative find cubes of variable valuations to the free variables that falsify the input formula. Such cubes are then added to the input formula in the next round of the algorithm by encoding these using the Enc^- function in line 2 of the algorithm.

3.4 Learning CDNF

In [9], Bshouty describes a learning algorithm for conjunctions of Boolean formulas in disjunctive normal form based on the *monotone theory*. Given some set of variables V, we call a Boolean function f over V c-monotone for some $c : V \to \mathbb{B}$ if $f' \circ m_c$ is monotone for m_c being the function mapping a variable valuation $x : V \to \mathbb{B}$ to some other valuation $x' : V \to \mathbb{B}$ such that for all $v \in V$: $x'(v) = x(v) \oplus c(v)$.

The main idea of Bshouty's learning algorithm is to represent the function to learn as a conjunction of Boolean formulas in disjunctive normal forms, where each of these formulas is c-monotone for some $c \in (V \to \mathbb{B})$. During the learning process, the algorithm maintains and updates the candidate CDNF formula ψ' learned so far. Whenever there exists a false-positive for this CDNF formula, i.e., there exists a valuation $f : (V \to \mathbb{B})$ for which $\psi'(f) = \mathbf{true}$ but $\psi(f) = \mathbf{false}$, a new DNF is added to ψ' that is kept f-monotone during the learning process. Whenever a false-negative is found for ψ', i.e., there exists a valuation $f : (V \to \mathbb{B})$ for which $\psi'(f) = \mathbf{false}$ but $\psi(f) = \mathbf{true}$, for every DNF ρ that is a conjunct of ψ' and its associated monotonicity base c, ρ is extended by some prime implicant for the c-monotone closure of ψ that is implied by f. For more details on Bshouty's CDNF learning algorithm, the interested reader is referred to [22].

Algorithm 3 shows the overall learning algorithm, adapted to the treatment of open QBF formulas. During its run, the set C contains the CDNF learned so far, split up into its DNF formulas, which are paired together with the respective monotonicity base.

In this algorithm, the function Enc^- is used to map a set C onto a CNF that encodes the negation of the CDNF formula represented by $C = \{(c_1, \rho_1), \ldots, (c_m, \rho_m)\}$, and EncV^- describes the necessary variables. We define:

$$\mathsf{EncV}^-(C) = \{v_i \mid 1 \leq i \leq m\}$$
$$\mathsf{Enc}^-(C) = \Big(\bigvee_{1 \leq i \leq m} v_i \Big) \wedge \bigwedge_{1 \leq i \leq m} (\neg v_i \vee \neg \rho_i)$$

Likewise, the function Enc^+ is used to map a set C onto a CNF that encodes it in non-negated form using the Tseitin encoding [23] and EncV^+ represents the necessary variables. The number of variables needed here is higher than for EncV^-, as one variable is introduced for every term in the DNFs of C.

In line 3 of the algorithm, it is checked whether C has a false-positive. A false-positive can be found by obtaining a satisfying assignment to the formula $\bigwedge_{(c,\rho) \in C} \rho$ $\wedge \neg \psi$. However, when ψ is in prenex-normal form, negating ψ requires a re-encoding to get $\neg \psi$ back into prenex-cnf. In Algorithm 3, this problem is circumvented by searching for a witness for the non-universality of $\bigvee_{(c,\rho) \in C} \neg \rho \vee \psi$ instead (which is the negation of the former formula).

Whenever a false-negative is found, all DNFs in C for which the false-negative is not a model are updated to change this fact. At the end of the algorithm (starting with line 22), redundant terms in the DNFs and redundant literals are identified using standard SAT solving (lines 26 and 32), and then removed.

4 QBF Solver Modification

All operations on open QBF formulas can easily be translated to the closed QBF case as follows: checking for satisfiability of an open QBF formula ψ over V amounts to testing if $\exists V.\psi \equiv \mathbf{true}$, and checking the non-universality of an open QBF formula ψ amounts to testing if $\forall V.\psi \equiv \mathbf{false}$. In the former case, the solver must be able to output a partial model of ψ. This is supported by many modern QBF solvers when the outermost quantification level is existentially quantified, which is the case for $\exists V.\psi \equiv \mathbf{true}$. For the non-universal (unsatisfiable) case, where the outermost quantification level is universal ($\forall V.\psi \equiv \mathbf{false}$), our QBF solver oracle needs to output a trace, or path, that leads to an unsatisfiable branch of the search space. This is referred to here as a co-model.

In this work we use the QBF solver QuBE 7.2 [15]. QuBE is a state-of-the-art DPLL search-based solver designed to take closed formulas as input, where $V = \emptyset$. This obstacle is circumvented by making free variables quantified as described above, and

Algorithm 3: CDNF learning using a QBF solver as oracle

```
Data: An open QBF instance $\psi = Q_1 x_1.Q_2 x_2.\dots Q_n x_n.\phi$ over the set of variables $V$
Result: The set of its models represented as CDNF $\psi'$
begin
1    $C := \emptyset$
2    while true do
3        if $\rho := \exists s, \mathsf{EncV}^-(C).Q_1 x_1.Q_2 x_2.\dots Q_n x_n.((\phi \vee s) \wedge (\mathsf{Enc}^-(C) \vee \neg s))$ is non-universal then
4            $f$ = co-model of $\rho$
5            $C := C \cup \{(f, \mathbf{false})\}$
6        if $\rho := \exists \mathsf{EncV}^-(C).Q_1 x_1.Q_2 x_2.\dots Q_n x_n.\phi \wedge \mathsf{Enc}^-(C)$ is satisfiable then
7            $f$ = model of $\rho$
8            $C' = \emptyset$
9            for $(c, \eta) \in C$ do
10               if $f \models \eta$ then
11                   $C' = C' \cup \{(c, \eta)\}$
12               else
13                   for $v \in V$ do
14                       if $c(v) \neq f(v)$ then
15                           $f' = f \setminus (v \mapsto f(v)) \cup (v \mapsto c(v))$
16                           if $Q_1 x_1.Q_2 x_2.\dots Q_n x_n.\phi \wedge \mathrm{term}(f')$ is satisfiable then
17                               $f := f'$
18                   $\eta' = \left(\bigwedge_{v \in V, f(v)=\mathbf{true}, c(v)=\mathbf{false}} v\right) \wedge \left(\bigwedge_{v \in V, f(v)=\mathbf{false}, c(v)=\mathbf{true}} \neg v\right)$
19                   $C' = C' \cup \{(c, \eta \vee \eta')\}$
20           $C := C'$
21       else
22           for $(c, \eta) \in C$ do
23               for $t \in \mathrm{terms}(\eta)$ do
24                   $\eta' := \bigvee_{t' \in \mathrm{terms}(\eta) \setminus \{t\}} t'$
25                   $C' := C \setminus \{(c, \eta)\} \cup \{(c, \eta')\}$
26                   if $\exists V, \mathsf{EncV}^+(C), \mathsf{EncV}^-(C').(\mathsf{Enc}^+(C) \wedge \mathsf{Enc}^-(C')) \equiv$ false then
27                       $C := C' \cup \{(c, \eta')\}$
28                   else
29                       for $l \in \mathrm{lits}(t)$ do
30                           $\eta' := \bigvee_{t' \in \mathrm{terms}(\eta) \setminus \{t\}} t' \vee \bigwedge_{l' \in \mathrm{lits}(t) \setminus \{l\}} l'$
31                           $C' := C \setminus \{(c, \eta)\} \cup \{(c, \eta')\}$
32                           if $\exists V, \mathsf{EncV}^-(C), \mathsf{EncV}^+(C').(\mathsf{Enc}^-(C) \wedge \mathsf{Enc}^+(C')) \equiv$ false then
33                               $C := C' \cup \{(c, \eta')\}$
34           return $\psi' = \bigwedge_{(c,\eta) \in C} \rho$
end
```

pushing them to the outermost quantifier level. Furthermore, QuBE incorporates many modern techniques: with respect to this paper, it includes for instance pure/don't care literal detection [12], conflict and solution analysis with solution cube minimization and learning [13,14,20], and an advanced preprocessor [16] that allows it to achieve unmatched performance when compared to the pure search-based algorithm. However, many of these advanced techniques have to be modified in order to produce correct partial models and co-models of the input formula.

Both the preprocessor and the solver were modified in order to keep information on the outermost quantified variables in a slightly more advanced way than a plain use of *don't touch literals* techniques (also called *frozen literals*) as previously done, e.g. in [19]. Indeed, either existential or universal variables are selectively "not touched" according to the outermost binding quantifier.

4.1 Preprocessing Phase

The preprocessor must behave in a slightly different way depending on the quantifier at the highest level of the prefix of the input formula. In case this is an existential quantifier, no variable elimination nor variable renaming techniques —such as equivalence reasoning, variable elimination by Q-resolution, and the subsumption through resolution (self-subsumption) that are not model preserving transformations which can be applied to existential variables— are performed on don't touch variables. Rather, unit and pure literals can still be given a value, simply adding it to the model. In the second case, where the input formula is bound by a universal quantifier, no preprocessing techniques have to be deactivated. Basically, the only rules of inference normally applied to universal variables are pure literal detection and universal reduction, also known as clause minimization [18]. Universal pure literals are immediately pushed into the current model: Note, by the definition of universal pure literal in Section 2, that this valuation to the variable will be sound in order to falsify the clause in case the formula is unsatisfiable. The universal reduction rule states that in a clause, whose literals have been simplified according to their evaluations, the literals quantified at the innermost prefix level among all the others can be removed if universal. This operation is performed every time a clause is added to the matrix — for instance, when a resolvent computed in a Q-resolution step substitutes its two antecedents, and every time a clause is simplified (e.g. when in a clause the existential literal l is deleted because $\neg l$ is unit). Whenever a universal reduction rule is applied, we track the universal literals being deleted: if all the literals in the clause are eliminated, resulting in the empty clause that proves the unsatisfiability of the whole formula, those literals are pushed into the model with their sign flipped.

4.2 Search Phase

During the search, in order to extract the model we have to record the value given to the outermost quantified variables as soon as a conflict occurs or a solution is found. This is done selectively for conflicts if the outermost quantifier is existential, or for solutions when the outermost quantifier is universal. When the solving procedure completes the exploration of the search space, the assignment values saved previously can be eligible to be included into the model. Indeed, because of the on-the-fly universal reduction performed by the solver during both exploration and backtrack phases on the clauses (respectively, its dual existential reduction being performed on the solution cubes), it may be the case that some valuations must be changed, or even further variables must be pushed into the model as well. This can happen when their values have already been taken from the assignment stack and put into the model or no valuation is currently given. In these situations, the valuation must be either flipped in case it is currently satisfying the clause/cube, or forced in case it has no valuation yet. Consider the QBF $\varphi = \forall y_1 y_2 \exists x_3 \phi$ and the empty clause $y_1 \vee y_2 \vee x_3 \in \phi$. Assume that y_2 is a decision literal at level d, and $\neg x_3$ is assigned because of unit propagation at the same decision level d. No value was given to $|y_1|$. As soon as the conflict occurred, the assignment to the outermost quantified universal variables was cached as y_2. Since the empty clause has led the conflict analysis to the root of the search tree, witnessing the unsatisfiability of the whole QBF, the model has to be modified as follows: $\neg y_1$ is added, and the value for $|y_2|$ is flipped into $\neg y_2$.

5 Applications of ALLQBF solving

In this section, we sketch two applications of ALLQBF solving. The aim of this section is twofold: first of all, our experimental evaluation in the next section is based on benchmarks from these two applications. Second, we want to show the interested reader *why* the transition from plain QBF solving to ALLQBF solving is such an interesting one.

5.1 Synthesis of Reactive Systems

In formal verification, one analyses a system for correctness with respect to a specification after it has been designed. The idea behind *synthesis* is that we can actually construct a system directly from the specification, and save the manual work of actually designing it. After choosing a formal specification language, and describing which inputs and outputs the system under design has, synthesis is essentially a push-button technique.

On the technical level, synthesis is typically reduced to *solving a game with an ω-regular winning condition*. A play in this game represents a *trace* of the system to be synthesized, and plays that are winning for a designated *system* player in turn represent traces that are allowed by the specification. Winning plays are of infinite length, meaning that the system they represent has no predefined point of going out of service. Games frequently have huge state spaces, but are representable in a symbolic way. Determining the positions in the game from which the system player wins is typically done by evaluating a *fixed point* expression. For example, in case of a safety specification, the synthesis problem reduces to safety game solving, and the winning positions are the ones that are not in the *attractor* of the *bad positions*, i.e., the ones from which the system player can enforce never to visit any of the bad states. Using a quantified Boolean formula, we can represent the problem if for a position in game there exists an output such that for every input, a non-bad state is reached. By performing ALLQBF solving on this formula, we obtain a small representation of *all* of these positions if the game transition function has a small encoding as CNF formula. Then, we can plug in the negation of this positions set as the new set of bad states, and obtain a formula whose models represent the positions from which the system player does not lose in two steps. Iterating this idea until we reach a fixed point finally gets us the set of states that are winning for the system player.

Currently, synthesis tools use BDDs [10] or anti-chains [11] for symbolic reasoning, which are both techniques with well-known scalability limits. Plain QBF solving has been proposed earlier for *bounded* safety game solving [1], but was found to have a bad performance there. For the initial experimental evaluation in this paper, we used a modified version of the UNBEAST synthesis tool [10] that lazily tracks the operations performed on BDDs and generates QBF instances to represent pre-fixed points in the game solving fixed point computation when synthesizing a load balancer. As the computations performed by UNBEAST are optimized towards BDD usage, we refrain from declaring a "winner" in this setting.

5.2 Monitor Synthesis

Consider a safety-critical sequential circuit. Adding a *runtime monitor* for such a circuit, that observes the transitions in the system and produces a warning signal if the system is about to enter a bad state, allows warning other parts of the system that the output of this circuit should not be trusted any more. This way, malicious bit flips in the hardware, as well as assumptions about the system environment that actually do not hold in practice, can often be found even after the system is deployed.

ALLQBF solving can help us in constructing such a monitor. For some value of k, we encode the problem "for a given state in the system, along some trace of length k starting from there, we do not visit a bad state" into an open QBF formula, and leave the starting state variables open. A CDNF, DNF or CNF representation for all of these states can then be interpreted as a circuit that checks if the system is still in a state that is not potentially bad, and outputs false if this is not the case.

For our experimental evaluation in the next section, we used the single-property circuits from the hardware model checking competition HWMCC'11 [6] and translated the monitor synthesis problem for $k = 2$ into an open QBF formula using a standard Tseitin encoding.

6 Experimental Results

We evaluate a prototype implementation of the learning techniques presented in Section 3 using a version of QuBE 7.2 that has been modified as described in Section 4. Before learning, we simplify the open QBF instance by applying a restricted version of QuBE's preprocessor that does not alter its set of models.

The aim of this experimental evaluation is to show that the proposed approaches already scale to systems of practical size. Due to the fact that ALLQBF solving is a relatively new topic, comparing against other solvers is difficult. For example, the QBF solver QUANTOR [4] is based on removing quantifiers by resolution and expansion, and in principle it is possible to obtain a CNF representation of the set of models of an open QBF formula. However, it comes with no possibility to ensure that the open (or outermost existential) variables remain intact and thus cannot be used.Techniques for removing only existential variables cannot handle the universal ones dealt with here.

All computation times given in the following are obtained on a Sun XFire computer with 2.6 Ghz AMD Opteron processors running an x64-version of Linux. The memory usage was never observed to exceed 2GB.

6.1 Synthesis of Reactive Systems

Out of the 3411 game solving/synthesis benchmarks, the maximally allowed computation time of 3600 seconds was enough for CDNF learning to work in 3307 cases, CNF learning to finish in 3358 cases, and DNF learning to succeed in 3297 case. Figure 1 shows the numbers of instances learned over time, while Figure 2 compares the number of literals in the learned model set representation. Table 1 shows the properties of some example instances used in this comparison.

It can be seen that in many cases, the result sizes of the techniques coincide - then, DNF learning is often the fastest method. However, for complicated benchmarks, when granting more time for solving an instance, the CNF variant overtakes the DNF variant in terms of instances solved. The experiments show that the CDNF learning method is a reasonable compromise between the two.

Table 1: Properties of some example problem instances in the game solving benchmarks. For every instance, "# V.", "# F." and "# Clauses" denote the numbers of variables, free variables and clauses in the instance, respectively. The column "P.S.-time" contains the time for plain QBF solving the instance. For CDNF, DNF and CNF learning, the sizes (s.) of the resulting formulas and the time (t.) to obtain these are reported. All times are given in seconds.

Instance	# V.	# F.	# Cl.	P.S. -time	# Models	CDNF s.	CDNF t.	CNF s.	CNF t.	DNF s.	DNF t.
load_16.xml_SAT_4_10	1185	18	1349	0.04	90112	10	3.62	9	2.12	12	1.83
load_18.xml_SAT_4_10	1935	28	2179	0.06	9.22747e+07	10	7.02	9	3.90	12	3.05
load_31.xml_SAT_4_10	11299	108	13090	0.16	1.08939e+29	31	192	19	286	42	45.6
load_33.xml_SAT_5_7	3396	49	3588	0.08	1.5668e+13	14	21.4	21	21.2	31	6.90
load_35.xml_SAT_5_7	5401	75	5746	0.11	2.62866e+20	16	50.7	23	60.3	39	16.0
load_57.xml_UNSAT_5_6	2924	37	4330	0.05	unknown	timeout		timeout		timeout	
load_72.xml_UNSAT_4_3	1181	37	1470	0.01	3.70482e+09	timeout		109	645	timeout	
load_74.xml_SAT_2_8	10402	73	10115	0.14	2.95148e+20	6	49.6	6	69.3	6	8.09
load_75.xml_SAT_2_9	14390	88	14425	0.21	9.67141e+24	6	95.2	6	130	6	13.4

6.2 Monitor Synthesis

Out of the 465 monitor synthesis benchmarks, in 286 cases, the learning process did not finish for any mode within a time limit of 15 minutes. In 65 additional cases, the set of models was empty or contained all possible variable valuations (and is therefore uninteresting for the purpose of monitoring). Table 2 shows some representative remaining

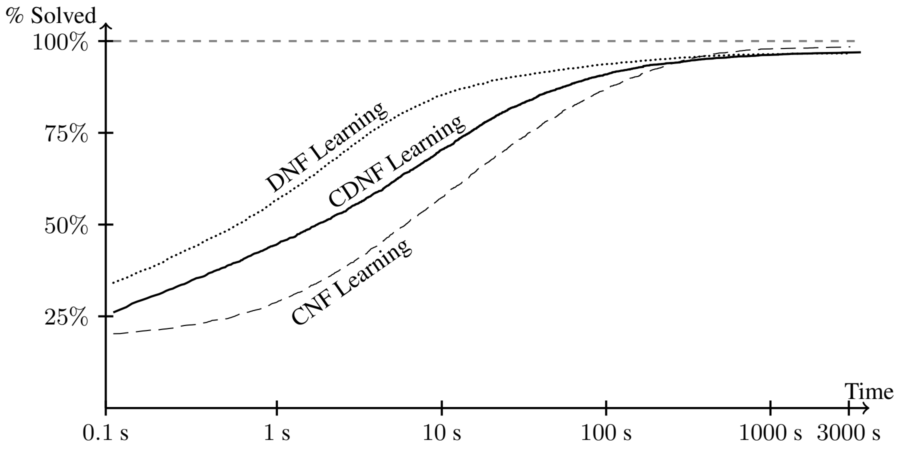

Fig. 1: Graph showing how many of the reactive synthesis benchmarks could be solved (i.e., the set of their models is learned) within certain time bounds.

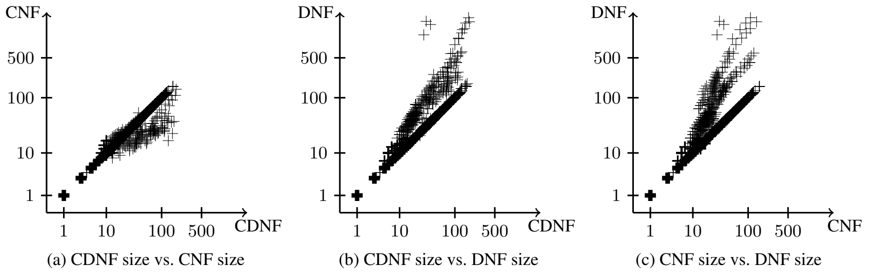

Fig. 2: Formula size comparisons of the learning results for CDNF, CNF, and DNF on the game solving benchmarks.

cases. Compared to the game solving benchmarks, it can be seen that the performance is worse, which is due to the fact that the monitor synthesis benchmarks are harder to solve: they have more variables, more clauses, and are derived from challenging hardware model checking problems.

For monitor synthesis, CDNF learning is not as competitive as in the game solving case, and DNF learning is more advisable to use here than CNF learning. Figure 3 shows the performance.

7 Conclusion

We have presented a way to turn a search-based QBF solver into an ALLQBF solver for open quantified Boolean formulas by using computational learning. The resulting set of models of a formula is represented either in DNF, CNF, or CDNF form, and we gave suitable learning algorithms for all of these forms.

Table 2: Properties of some example problem instances in the game solving benchmarks. The notation is the same as in Table 1.

Instance	**# V.**	**# F.**	**# Cl.**	**P.S. -time**	**# Models**	**CDNF s.**	**CDNF t.**	**CNF s.**	**CNF t.**	**DNF s.**	**DNF t.**
`bob3`	1232	74	3218	0.1	7.41919e+21	timeout		timeout		1211	485
`bob9234redmiter`	1843	119	4627	0.15	2.10288e+35	timeout		384	798	358	371
`irstdme4`	2534	124	6088	0.21	1.66153e+37	25	245	46	180	15	48.9
`pdtvisgigamax0`	2276	16	6566	0.21	64512	timeout		7	185	7	21.7
`vis4arbitp1`	747	23	2024	0.07	4.18867e+06	timeout		652	389	331	29.3

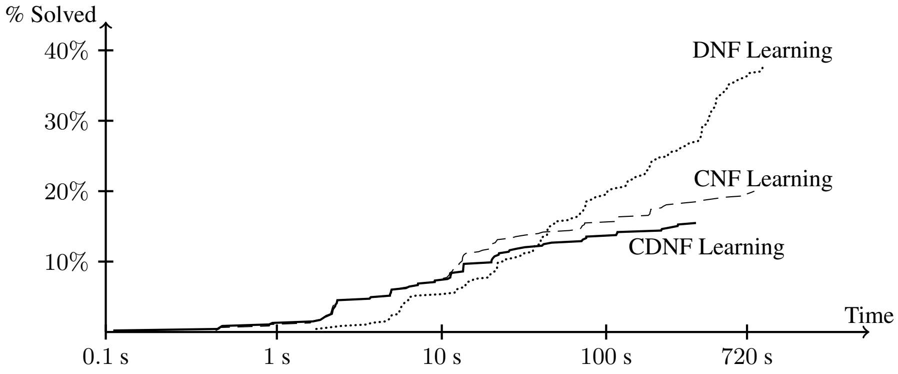

Fig. 3: Graph showing how many of the monitor synthesis benchmarks could be solved (i.e., the set of their models is learned) within certain time bounds.

The initial evaluation of the approach in this paper shows its potential. We conjecture that a future tighter integration of the solver and the learning algorithm will provide a significant further speed improvement.

References

1. Alur, R., Madhusudan, P., Nam, W.: Symbolic computational techniques for solving games. STTT 7(2), 118–128 (2005)
2. Angluin, D.: Queries and concept learning. Machine Learning 2(4), 319–342 (1987)
3. Benedetti, M., Mangassarian, H.: QBF-based formal verification: Experience and perspectives. JSAT 5(1-4), 133–191 (2008)
4. Biere, A.: Resolve and expand. In: Proc. SAT. pp. 59–70 (2004)
5. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, R. (ed.) TACAS. LNCS, vol. 1579, pp. 193–207. Springer (1999)
6. Biere, A., Heljanko, K., Wieringa, S., Sörensson, N.: 4th hardware model checking competition (HWCC'11). Affiliated with FMCAD'11, http://fmv.jku.at/hwmcc11/
7. Brauer, J., King, A., Kriener, J.: Existential quantification as incremental SAT. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV. LNCS, vol. 6806, pp. 191–207. Springer (2011)
8. Brauer, J., Simon, A.: Inferring definite counterexamples through under-approximation. In: NASA Formal Methods. LNCS, vol. 7226, pp. 54–69. Springer (2012)
9. Bshouty, N.H.: Exact learning Boolean function via the monotone theory. Inf. Comput. 123(1), 146–153 (1995)

10. Ehlers, R.: Unbeast: Symbolic bounded synthesis. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS. LNCS, vol. 6605, pp. 272–275. Springer (2011)
11. Filiot, E., Jin, N., Raskin, J.F.: An antichain algorithm for LTL realizability. In: Bouajjani, A., Maler, O. (eds.) CAV. LNCS, vol. 5643, pp. 263–277. Springer (2009)
12. Gent, I., Giunchiglia, E., Narizzano, M., Rowley, A., Tacchella, A.: Watched data structures for QBF solvers. In: Giunchiglia, E., Tacchella, A. (eds.) SAT. LNCS, vol. 2919, pp. 25–36. Springer (2004)
13. Gent, I.P., Rowley, A.G.D.: Solution Learning and Solution Directed Backjumping Revisited. In: Technical Report APES-80-2004, APES Research Group (2004)
14. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/term resolution and learning in the evaluation of quantified Boolean formulas. Journal of Artificial Intelligence Research (JAIR) 26, 371–416 (2006)
15. Giunchiglia, E., Marin, P., Narizzano, M.: QuBE7.0, System Description. JSAT 7(8), 83–88 (2010)
16. Giunchiglia, E., Marin, P., Narizzano, M.: sQueezeBF: An effective preprocessor for QBFs. In: Theory and Applications of Satisfiability Testing. Springer Verlag (2010), LNCS
17. Hellerstein, L., Raghavan, V.: Exact learning of DNF formulas using DNF hypotheses. J. Comput. Syst. Sci. 70(4), 435–470 (2005)
18. Kleine-Büning, H., Karpinski, M., Flögel, A.: Resolution for quantified Boolean formulas. Information and Computation 117(1), 12–18 (1995)
19. Kupferschmid, S., Lewis, M., Schubert, T., Becker, B.: Incremental preprocessing methods for use in BMC. In: Int'l Workshop on Hardware Verification (July 2010)
20. Marin, P., Giunchiglia, E., Narizzano, M.: Conflict and solution driven constraint learning in QBF. In: Doctoral Program of Constraint Programming Conference 2010 (CP 2010) (2010)
21. McMillan, K.L.: Applying SAT methods in unbounded symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV. LNCS, vol. 2404, pp. 250–264. Springer (2002)
22. Sloan, R.H., Szörényi, B., Turán, G.: Learning Boolean functions with queries. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press (2010)
23. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic, Part 2 pp. 115–125 (1970)
24. Valiant, L.G.: A theory of the learnable. Commun. ACM 27(11), 1134–1142 (1984)